SPIDER-DAY07

1. 腾讯招聘爬虫

- **scrapy项目代码**

```
见day07笔记：Tencent 文件夹
【1】一级页面
    提取数据：每个职位的PostId
【2】二级页面
    提取数据：1个职位的 名称、地点、类别、发布时间、职责、要求
```

2. 腾讯招聘数据持久化

- **建库建表SQL**

```
create database tencentdb charset utf8;
use tencentdb;
create table tencenttab(
job_name varchar(200),
job_type varchar(200),
job_duty varchar(2000),
job_require varchar(2000),
job_add varchar(100),
job_time varchar(100)
)charset=utf8;
```

- **MySQL数据持久化实现**

```
【1】pipelines.py新建MySQL管道类
import pymysql

class TencentMysqlPipeline:
    def open_spider(self, spider):
        self.db = pymysql.connect('localhost', 'root', '123456', 'tencentdb', charset='utf8')
        self.cur = self.db.cursor()
        self.ins = 'insert into tencenttab values(%s,%s,%s,%s,%s,%s)'

    def process_item(self, item, spider):
        li = [
            item['job_name'],
            item['job_type'],
```

```
            item['job_duty'],
            item['job_require'],
            item['job_add'],
            item['job_time'],
        ]
        self.cur.execute(self.ins, li)
        self.db.commit()

        return item

    def close_spider(self, spider):
        self.cur.close()
        self.db.close()

【2】settings.py添加
ITEM_PIPELINES = {
   # 在原来基础上添加MySQL的管道
   'Tencent.pipelines.TencentMysqlPipeline': 200,
}
```

- **MongoDB数据持久化实现**

```
【1】pipelines.py中新建MongoDB管道类
import pymongo

class TencentMongoPipeline:
    def open_spider(self, spider):
        self.conn = pymongo.MongoClient('localhost', 27017)
        self.db = self.conn['tencentdb']
        self.myset = self.db['tencentset']

    def process_item(self, item, spider):
        self.myset.insert_one(dict(item))

【2】settings.py中添加
ITEM_PIPELINES = {
   # 添加MongoDB管道
   'Tencent.pipelines.TencentMongoPipeline': 400,
}
```

- **csv及json数据持久化实现**

```
【1】csv
   scrapy crawl tencent -o tencent.csv

【2】json
   settings.py中添加变量: FEED_EXPORT_ENCODING = 'utf-8'
   scrapy crawl tencent -o tencent.json
```

3. scrapy.Request()参数

```
【1】url         : 指定URL地址
【2】callback    : 指定解析函数
【3】meta={}     : 不同解析函数间传递数据
【4】dont_filter: 是否参与调度器的去重,默认为False,设置为True不去重
```

4. 分布式爬虫

4.1 分布式爬虫概述

```
【1】原理
    多台主机共享1个爬取队列

【2】实现
    2.1) 重写scrapy调度器(scrapy_redis模块)
    2.2) sudo pip3 install scrapy_redis

【3】为什么使用redis
    3.1》Redis基于内存,速度快
    3.2》Redis非关系型数据库,Redis中集合,存储每个request的指纹
```

4.2 scrapy_redis详解

- **GitHub地址**

```
https://github.com/rmax/scrapy-redis
```

- **settings.py说明**

```
# 重新指定调度器: 启用Redis调度存储请求队列
SCHEDULER = "scrapy_redis.scheduler.Scheduler"

# 重新指定去重机制: 确保所有的爬虫通过Redis去重
DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"

# 不清除Redis队列: 暂停/恢复/断点续爬(默认清除为False,设置为True不清除)
SCHEDULER_PERSIST = True

# redis管道
ITEM_PIPELINES = {
    'scrapy_redis.pipelines.RedisPipeline': 300
}

#指定连接到redis时使用的端口和地址
REDIS_HOST = 'localhost'
REDIS_PORT = 6379
```

4.3 腾讯招聘分布式爬虫

- **分布式爬虫完成步骤**

```
【1】首先完成非分布式scrapy爬虫 ： 正常scrapy爬虫项目抓取
【2】设置,部署成为分布式爬虫
```

- **分布式环境说明**

```
【1】分布式爬虫服务器数量: 2 (其中1台Windows,1台Ubuntu虚拟机)
【2】服务器分工:
    2.1) Windows : 负责数据抓取
    2.2) Ubuntu  : 负责URL地址统一管理,同时负责数据抓取
```

- **腾讯招聘分布式爬虫 - 数据同时存入1个Redis数据库**

```
【1】完成正常scrapy项目数据抓取 (非分布式 - 拷贝之前的Tencent)

【2】设置settings.py, 完成分布式设置
    2.1-必须) 使用scrapy_redis的调度器
        SCHEDULER = "scrapy_redis.scheduler.Scheduler"

    2.2-必须) 使用scrapy_redis的去重机制
        DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"

    2.3-必须) 定义redis主机地址和端口号
        REDIS_HOST = '192.168.1.107'
        REDIS_PORT = 6379

    2.4-非必须) 是否清除请求指纹,True:不清除 False:清除 (默认)
        SCHEDULER_PERSIST = True

    2.5-非必须) 在ITEM_PIPELINES中添加redis管道,数据将会存入redis数据库
        'scrapy_redis.pipelines.RedisPipeline': 200

【3】把代码原封不动的拷贝到分布式中的其他爬虫服务器,同时开始运行爬虫

【结果】: 多台机器同时抓取,数据会统一存到Ubuntu的redis中, 而且所抓数据不重复
```

- **腾讯招聘分布式爬虫 - 数据存入MySQL数据库**

```
"""和数据存入redis步骤基本一样,只是变更一下管道和MySQL数据库服务器的IP地址"""
【1】settings.py
    1.1) SCHEDULER = 'scrapy_redis.scheduler.Scheduler'
    1.2) DUPEFILTER_CLASS = 'scrapy_redis.dupefilter.RFPDupeFilter'
    1.3) SCHEDULER_PERSIST = True
    1.4) REDIS_HOST = '192.168.1.105'
    1.5) REDIS_PORT = 6379
    1.6) ITEM_PIPELINES = {'Tencent.pipelines.TencentMysqlPipeline' : 300}

【2】将代码拷贝到分布式中所有爬虫服务器

【3】多台爬虫服务器同时运行scrapy爬虫
```

```

# 赠送腾讯MySQL数据库建库建表语句
"""
create database tencentdb charset utf8;
use tencentdb;
create table tencenttab(
job_name varchar(1000),
job_type varchar(200),
job_duty varchar(5000),
job_require varchar(5000),
job_address varchar(200),
job_time varchar(200)
)charset=utf8;
"""
```

5. 图形验证码处理

5.1 机器视觉概述

```
【1】作用
    处理图形验证码

【2】三个重要概念 - OCR、tesseract-ocr、pytesseract
    2.1) OCR
        光学字符识别(Optical Character Recognition),通过扫描等光学输入方式将各种票据、报刊、书籍、文稿及其它印刷品的文字转化为图像信息，再利用文字识别技术将图像信息转化为电子文本

    2.2) tesseract-ocr
        OCR的一个底层识别库（不是模块，不能导入），由Google维护的开源OCR识别库

    2.3) pytesseract
        Python模块,可调用底层识别库，是对tesseract-ocr做的一层Python API封装
```

5.2 安装

- **安装tesseract-ocr**

```
【1】安装tesseract-ocr
    1.1》Ubuntu安装 ： sudo apt-get install tesseract-ocr
    1.2》Windows安装
        下载安装包，双击安装
        添加到环境变量(Path)
【2】测试（终端 | cmd命令行）
    tesseract xxx.jpg 文件名
```

- **安装pytesseract**

```
【1】安装
    sudo pip3 install pytesseract

【2】使用示例
    import pytesseract
    # Python图片处理库
    from PIL import Image

    # 创建图片对象
    img = Image.open('test1.jpg')
    # 图片转字符串
    result = pytesseract.image_to_string(img)
    print(result)
```

- **面试问题: 如何处理爬虫中遇到的验证码**

```
【1】图形验证码
    简单的图形验证码,我使用tesseract-ocr去处理
    对于一些复杂的验证码,我们使用在线打码(图鉴、云打码)
【2】滑块、缺口验证码
    使用selenium处理
    或者使用人工打码
```

6. 滑块缺口验证码

6.1 豆瓣网登录爬虫

6.1.1 项目需求

```
【1】URL地址: https://www.douban.com/
【2】先输入几次错误的密码，让登录出现滑块缺口验证，以便于我们破解
【3】模拟人的行为(总距离: 200)
    3.1) 先快速滑动一部分距离(滑动160)
    3.2) 剩余距离(40): 先匀加速(40*4/5=32),再匀减速(40*1/5=8)
【4】详细看代码注释
```

6.1.2 项目实现

```
"""
说明: 先输入几次错误的密码，出现滑块缺口验证码
"""
from selenium import webdriver
# 导入鼠标事件类
from selenium.webdriver import ActionChains
import time

# 加速度函数
def get_tracks(distance):
    """
```

```
    拿到移动轨迹，模仿人的滑动行为，先匀加速后匀减速
    匀变速运动基本公式：
    ①v=v0+at
    ②s=v0t+½at²
    """
    # 初速度
    v = 0
    # 单位时间为0.3s来统计轨迹，轨迹即0.3内的位移
    t = 0.3
    # 位置/轨迹列表,列表内的一个元素代表0.3s的位移
    tracks = []
    # 当前的位移
    current = 0
    # 到达mid值开始减速
    mid = distance*4/5
    while current < distance:
        if current < mid:
            # 加速度越小,单位时间内的位移越小,模拟的轨迹就越多越详细
            a = 2
        else:
            a = -3

        # 初速度
        v0 = v
        # 0.3秒内的位移
        s = v0*t+0.5*a*(t**2)
        # 当前的位置
        current += s
        # 添加到轨迹列表
        tracks.append(round(s))
        # 速度已经达到v，该速度作为下次的初速度
        v = v0 + a*t
    return tracks
    # tracks: [第一个0.3秒的移动距离,第二个0.3秒的移动距离,...]

# 1、打开豆瓣官网 - 并将窗口最大化
driver = webdriver.Chrome()
driver.get(url='https://www.douban.com/')
driver.maximize_window()

# 2、切换到iframe子页面
iframe_node = driver.find_element_by_xpath('//div[@class="login"]/iframe')
driver.switch_to.frame(iframe_node)
# 3、密码登录 + 用户名 + 密码 + 登录豆瓣
driver.find_element_by_xpath('/html/body/div[1]/div[1]/ul[1]/li[2]').click()
driver.find_element_by_xpath('//*[@id="username"]').send_keys('15110225726')
driver.find_element_by_xpath('//*[@id="password"]').send_keys('aaa')

while True:
    try:
        driver.find_element_by_xpath('/html/body/div[1]/div[2]/div[1]/div[5]/a').click()
        time.sleep(3)

        # 4、切换到新的iframe子页面 - 滑块验证
        driver.switch_to.frame('tcaptcha_iframe')

        # 5、按住开始滑动位置按钮 - 先移动180个像素
```

```
        node = driver.find_element_by_xpath('//*[@id="tcaptcha_drag_button"]')
        # click_and_hold(): 鼠标按住某个节点并保持
        ActionChains(driver).click_and_hold(node).perform()
        # 鼠标移动到距离某个节点水平 及 垂直的距离
        ActionChains(driver).move_to_element_with_offset(to_element=node, xoffset=180,
yoffset=0).perform()
        # 6、使用加速度函数移动剩下的距离-25个像素
        # tracks: []
        tracks = get_tracks(25)
        for track in tracks:
            # 鼠标移动到距离当前位置水平 及 垂直的距离
            ActionChains(driver).move_by_offset(xoffset=track, yoffset=0).perform()

        # 7、延迟释放鼠标: release()
        time.sleep(0.5)
        # release(): 释放鼠标
        ActionChains(driver).release().perform()
    except:
        pass
```

7. Fiddler抓包工具

7.1 配置抓取浏览器数据包

- **配置Fiddler**

```
【1】Tools -> Options -> HTTPS
    1.1) 添加证书信任:  勾选 Decrypt Https Traffic 后弹出窗口, 一路确认
    1.2) 设置只抓浏览器的包:  ...from browsers only

【2】Tools -> Options -> Connections
    2.1) 设置监听端口 (默认为8888)

【3】配置完成后重启Fiddler ('重要')
    3.1) 关闭Fiddler,再打开Fiddler
```

- **配置浏览器代理**

```
【1】安装Proxy SwitchyOmega谷歌浏览器插件

【2】配置代理
    2.1) 点击浏览器右上角插件SwitchyOmega -> 选项 -> 新建情景模式 -> myproxy(名字) -> 创建
    2.2) 输入  HTTP://  127.0.0.1  8888
    2.3) 点击 : 应用选项

【3】点击右上角SwitchyOmega可切换代理

【注意】: 一旦切换了自己创建的代理,则必须要打开Fiddler才可以上网
```

7.2 Fiddler使用说明

- **Fiddler常用菜单**

```
【1】Inspector ： 查看数据包详细内容
    1.1) 整体分为请求和响应两部分

【2】Inspector常用菜单
    2.1) Headers ： 请求头信息
    2.2) WebForms: POST请求Form表单数据 ： <body>
                   GET请求查询参数: <QueryString>
    2.3) Raw ： 将整个请求显示为纯文本
```

8. 移动端app数据抓取

8.1 方式一(F12模拟)

8.1.1 有道翻译手机版爬虫

```
import requests
from lxml import etree

word = input('请输入要翻译的单词:')

post_url = 'http://m.youdao.com/translate'
post_data = {
  'inputtext':word,
  'type':'AUTO'
}

html = requests.post(url=post_url,data=post_data).text
parse_html = etree.HTML(html)
xpath_bds = '//ul[@id="translateResult"]/li/text()'
result = parse_html.xpath(xpath_bds)[0]

print(result)
```

8.2 方式二(手机&Fiddler)

```
设置方法见文件夹 - 移动端抓包配置
```